

REMARKS

Claims 1-30 are pending in the above-identified application. Claims 1, 15 and 27 are independent.

Applicant acknowledges with thanks the examiner's withdrawal of the previous objections to claims 10-12, and the withdrawal of the previous rejections of claims 27-30 under 35 U.S.C. §101.

The examiner rejected claims 1-8, 10-20 and 22-30 under 35 U.S.C. §102(b) as being anticipated by U.S. Pre-Grant Publication No. 2004/0006667 to Bik et al. The examiner further rejected claims 9 and 21 under 35 U.S.C. 103(a) as being unpatentable over Bik in view of the Microsoft Press Computer Dictionary.

Specifically, regarding independent claim 1, the examiner stated:

3. With respect to claim 1, Bik teaches a method comprising:
 - converting memory access instructions in a source code into standard formatted memory access instructions, in par. 31.
 - generating partitions containing the standard formatted memory access instructions, in par. 31.
 - generating a match set, the match set including matches of instruction patterns to the standard formatted memory access instructions in the partitions, in par. 31; and
 - transforming the matches to vector memory access instructions, in par. 31.

Applicant respectfully disagrees.

Applicant's independent claim 1 recites "converting memory access instructions in a source code into standard formatted memory access instructions; generating partitions containing the standard formatted memory access instructions; generating a match set, the match set including matches of instruction patterns to the standard formatted memory access instructions in the partitions; and transforming the matches to vector memory access instructions."

As described in the originally filed application, applicant's method involves:

the compiler 18 transforms (104) as many formatted memory access instructions as possible to a standard format in which the memory address reference is in the form of base address plus offset, where base address is an address that serves as a reference point for other addresses. For example, a base address can indicate the beginning of a program. The

address of every instruction in the program can then be specified by adding an offset to the base address. Transforming a memory access instruction to the form of base address plus offset facilitates memory access vectorization by making it easier to combine several memory accesses together.

After the pass of transforming memory access instructions the compiler 18 generates (106) memory access partitions for each set of memory reads or memory writes to a particular memory bank based on a data flow graph. A memory access partition contains groups of memory access instructions inside one basic block. All memory access instructions inside a group (also called subnode of a memory access partition) perform the same memory access (write or read) to the same memory bank. The compiler 18 vectorizes (108) multiple memory access instructions based on instruction pattern matching performed for each group (subnode) of each memory access partition. (page 3, line 21, to page 4, line 11 of the originally filed application)

Applicant's method, as recited in claim 1, converts the source-code memory access instructions into a standard format. In one embodiment (as recited, for example, in claim 3, and independent claim 15), that standard format includes a base address plus an offset. This standard format subsequently enables applicant's method to identify memory access instructions that access memory locations that are proximate to each other. Applicant's method also generates partitions into which the standard format instructions are grouped. The generation of partitions enables the applicant's method to identify suitable memory-access instruction candidates that potentially could be combined to yield a resultant single vectorized memory-access instruction.

Bik describes a method and apparatus for implementing adjacent, non-unit stride memory access patterns using single instruction, multiple data (SIMD) instructions (Bik, page 1, paragraph 1). As Bik explains:

[0031] A method and apparatus for implementing adjacent, non-unit stride memory access patterns utilizing SIMD instructions are described. In one embodiment, the method includes compiler analysis of a source program to detect vectorizable loops having serial code statements that collectively perform adjacent, non-unit stride memory access. Once a vectorizable loop containing code statements that collectively perform adjacent, non-unit stride memory access is detected, the system compiler vectorizes the serial code statements of the detected loop to perform the adjacent, non-unit stride memory access utilizing SIMD instructions. As such, the compiler repeats the analysis and vectorization for each vectorizable loop within the source program code. (emphasis added, Bik, page 2, paragraph 31)

Bik further explains that:

[0094] As described above, vectorizable loops refer to loops containing serial code statements that can be replaced with SIMD instruction statements to perform parallel processing of data elements. Accordingly, at process block 604, it is determined whether collective unit-stride memory access is detected while analyzing the source program. When the system compiler detects serial code statements that collectively perform unit-stride memory access, process block 660 is performed. At process block 660, the system compiler vectorizes serial code statements of each detected loop to perform adjacent, non-unit stride memory access, utilizing SIMD instructions ("SIMD vectorization"). (Bik, page 9, paragraph 94)

Thus, Bik's compiler detects loops having code sequences that perform repeated memory access operations to or from near-adjacent memory locations. Once those loops are detected, Bik's compiler vectorized those vectorizable loops into code statements that use SIMD instructions. Examples of the vectorizable loops that Bik's compiler detects and then transforms into code sequences that use SIMD instruction are provided, for example, in Tables 3, 5, 7 and 9 of the Bik reference.

However, while Bik describes that its compiler detects loops that perform adjacent, non-unit stride memory-access operations, and transforms them into SIMD instructions, at no point does Bik describe that its compiler converts the source-code memory-access instruction into standard format access instructions. Indeed, such a conversion of memory-access instructions would be entirely unnecessary for Bik's compiler because Bik's compiler detects loops whose corresponding memory access instructions are already known to access nearby memory locations. Therefore, there is no need to convert those instructions into a format that would subsequently enable determining whether the instructions perform memory-access operations to nearby memory locations. Although Bik describes at FIG. 13 and at paragraph 99 that "[a]t process block 642, a system compiler generates an internal representation of the source program code to enable vectorization analysis of serial code within the source program," this internal representation is applied to the entire source code to facilitate operation of the compiler, and there is no indication that this internal representation involves conversion of the memory-access instruction to some standard formatted memory-access instructions.

Additionally, Bik also fails to describe that partitions containing any type of instructions are generated. The generation of such partitions would be also entirely unnecessary because, as

noted above, the vectorizable loops that Bik's compiler detects are already known to include memory-access instructions of a particular type (e.g., store or load) to adjacent (specifically, non-unit stride adjacent) memory locations. Accordingly, no partitioning of instructions to enable identification of suitable memory-access instruction candidates for vectorization is needed.

Thus, Bik neither discloses nor suggests at least the features of "converting memory access instructions in a source code into standard formatted memory access instructions; generating partitions containing the standard formatted memory access instructions," as required by applicant's independent claim 1. Applicant's independent claim 1 is therefore patentable over the cited art.

Claims 2-14 depend from independent claim 1 and are therefore patentable for at least the same reasons as independent claim 1.

Independent claims 15 recites "converting the memory access instructions into a format including a base address plus an offset; grouping subsets of the converted memory access instructions into partitions."

The examiner stated with respect to claim 15:

16. With respect to claim 15, Bik teaches a compilation method comprising:
 - converting memory access instructions that read or write less than a minimum data access unit (MDAU) to memory access instructions that read or write a multiple of the minimum data access unit, in par. 4.
 - converting the memory access instructions into a format including a base address plus an offset, in par. 59, with reference to figure 3A.
 - grouping subsets of the converted memory access instructions into partitions, in par. 31; and
 - vectorizing the converted memory access instructions in the subsets that match instruction patterns, in par. 31.

Applicant respectfully disagrees.

Applicant's method as recited in independent claim 15 converts source-code memory-access instructions into a format that includes a base address and an offset. This format enable applicant's method to identify those instructions that perform memory access instruction to memory locations that are proximate to each other.

In contrast, and as explained above with respect to independent claim 1, Bik does not need to convert memory-access instructions to a standard format because the vectorizable loops Bik's compiler detects include instructions that are already known to perform memory access instruction to adjacent memory locations. Therefore, Bik fails to disclose converting memory access instruction into a standard format, and Bik certainly does not disclose "converting memory access instructions into a format including a base address plus an offset," as required by applicant's independent claim 15.

As for the examiner's contention that Bik's FIG. 3A and paragraph 59 disclose this feature, paragraphs 58-59 state:

[0058] Referring now to FIGS. 3A and 3B, FIGS. 3A and 3B illustrate 128-bit SIMD data type according to one embodiment of the present invention. FIG. 3A illustrates four 128-bit packed data-types 220, packed byte 222, packed word 224, packed doubleword (dword) 226 and packed quadword 228. Packed byte 222 is one hundred twenty-eight bits long containing sixteen packed byte data elements. Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. In packed data sequences, the number of data elements stored in a register is one hundred twenty-eight bits divided by the length in bits of a data element.

[0059] Packed word 224 is one hundred twenty-eight bits long and contains eight packed word data elements. Each packed word contains sixteen bits of information. Packed doubleword 226 is one hundred twenty-eight bits long and contains four packed doubleword data elements. Each packed doubleword data element contains thirty-two bits of information. A packed quadword 228 is one hundred twenty-eight bits long and contains two packed quad-word data elements. Thus, all available bits are used in the register. This storage arrangement increases the storage efficiency of the processor. Moreover, with multiple data elements accessed simultaneously, one operation can now be performed on multiple data elements simultaneously.

Applicant fails to see where in FIG. 3 and/or paragraph 59 there is disclosed "converting memory access instructions into a format including a base address plus an offset."

Additionally, as explained above with respect to independent claim 1, Bik does not need to partition instructions into groups so as to enable identifying suitable candidate instructions that could be combined to a single vectorized instruction, because the vectorizable loops that Bik's

compiler detects are already known to include memory-access instructions of a particular type (e.g., store or load) to adjacent memory locations.

Thus, Bik fails to disclose or suggest at least the features of “converting the memory access instructions into a format including a base address plus an offset; grouping subsets of the converted memory access instructions into partitions,” as required by applicant’s independent claim 15.

Claims 16-26 depend from independent claim 15 and are therefore patentable for at least the same reasons as independent claim 15.

Independent claim 27 recites “convert memory access instructions residing in a source code into standard formatted memory access instructions; generate partitions containing the standard formatted memory access instructions.” For at least similar reasons as those provided with respect to independent claim 1, at least these features are not disclosed by the cited art. Accordingly, independent claim 27 is patentable over the cited art.

Claims 28-30 depend from independent claim 27 and are therefore patentable for at least the same reasons as independent claim 27.

It is believed that all the rejections and/or objections raised by the examiner have been addressed.

In view of the foregoing, applicant respectfully submits that the application is in condition for allowance and such action is respectfully requested at the examiner’s earliest convenience.

All of the dependent claims are patentable for at least the reasons for which the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.

Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims, or (c) amended or canceled a claim

Applicant : Bo Huang et al.
Serial No. : 10/718,283
Filed : November 19, 2003
Page : 8 of 8

Attorney's Docket No.: 10559-886001 / Intel Corporation P17581

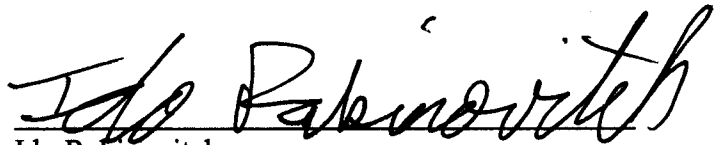
does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

No fee is believed due. Please apply any other charges or credits to deposit account 06-1050, referencing attorney docket 10559-886001.

Respectfully submitted,

Date:

Aug. 14, 2006

A handwritten signature in black ink, reading "Ido Rabinovitch". The signature is written in a cursive, flowing style. The first name "Ido" is written with a large, stylized 'I' and 'd'. The last name "Rabinovitch" is written with a large, stylized 'R' and 'v'.

Ido Rabinovitch

Attorney for Intel Corporation

Reg. No. L0080

PTO Customer No. 20985
Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906